

DOCUMENTATIE

Tema numărul 3
ORDERS MANAGEMENT

NUME STUDENT: Elecfi Sergiu-Andrei
GRUPA: 30223

CUPRINS

1.	Obiectivul temei.....	3
2.	Analiza problemei, modelare, scenarii, cazuri de utilizare	3
3.	Proiectare.....	4
4.	Implementare.....	4
5.	Rezultate.....	9
6.	Concluzii	9
7.	Bibliografie.....	9

1. Obiectivul temei

Cerința temei a fost : "Luați în considerare o aplicație "Order management" pentru procesarea comenzilor clienților pentru un depozit. Bazele de date relaționale ar trebui folosite pentru a stoca produsele, clienții și comenzile.

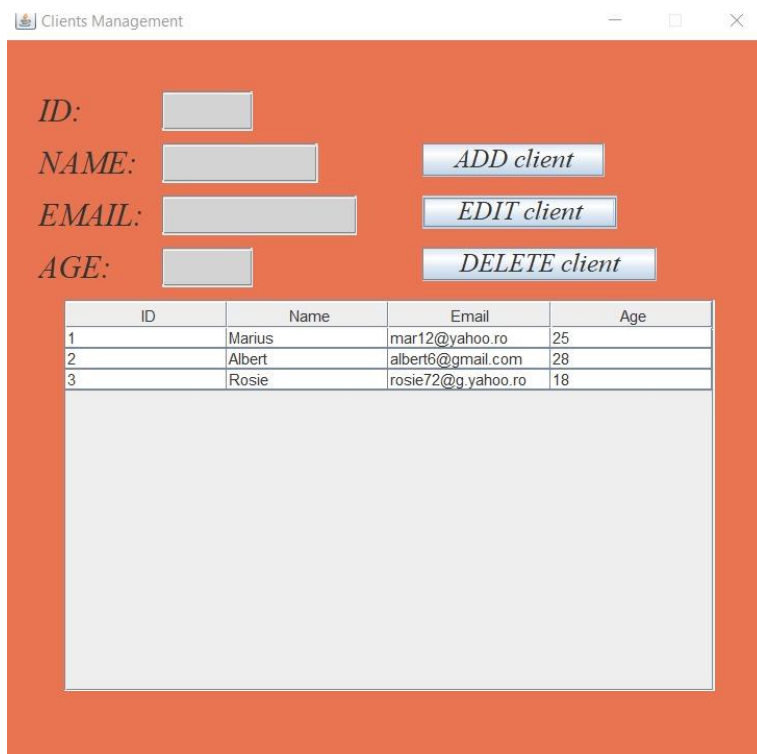
Aplicația ar trebui să fie proiectată conform modelului de arhitectură stratificată și ar trebui să fie utilizată (minim) următoarele clase:

- ➔ Model classes – reprezintă modelul de date al aplicației
- ➔ Business logic classes – conțin logica aplicației
- ➔ Presentation classes – clasele de tip GUI
- ➔ Data access classes – clasele ce contin accesul si operațiile asupra bazelor de date

2. Analiza problemei, modelare, scenarii, cazuri de utilizare

Proiectul este funcțional în totalitate. User – ul are posibilitățile să creeze comenzi, să vadă "chitanța" pentru fiecare comandă creată. De asemenea, user-ul are posibilitatea de a vedea live conținutul tabelelor "clients" și "products". La fiecare update, acestea se modifică și valorile vor apărea schimbate în tabela de afișare a acestora (exemplu dacă se șterge un produs, tabela de afișare se va modifica și vor apărea produsele rămase în magazin).

Mai mult, user – ul are posibilitatea, de a "adăuga", "șterge" și "edita" orice produs sau client. Am afișat tabelele dedesupt pentru a fi mai ușor să se vadă datele și să se creeze comenzi.



Clients Management

ID:

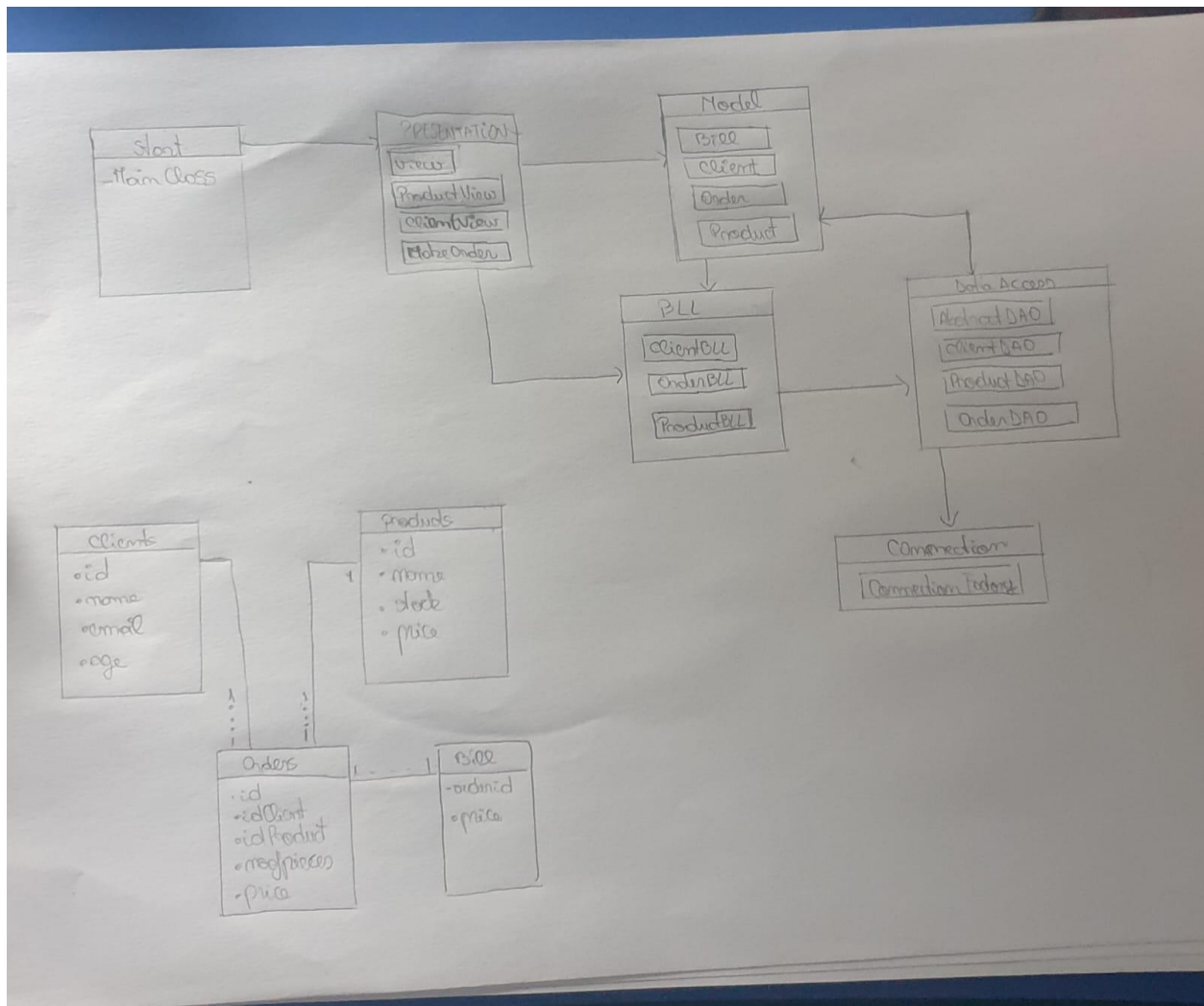
NAME:

EMAIL:

AGE:

ID	Name	Email	Age
1	Marius	mar12@yahoo.ro	25
2	Albert	albert6@gmail.com	28
3	Rosie	rosie72@g.yahoo.ro	18

3. Proiectare

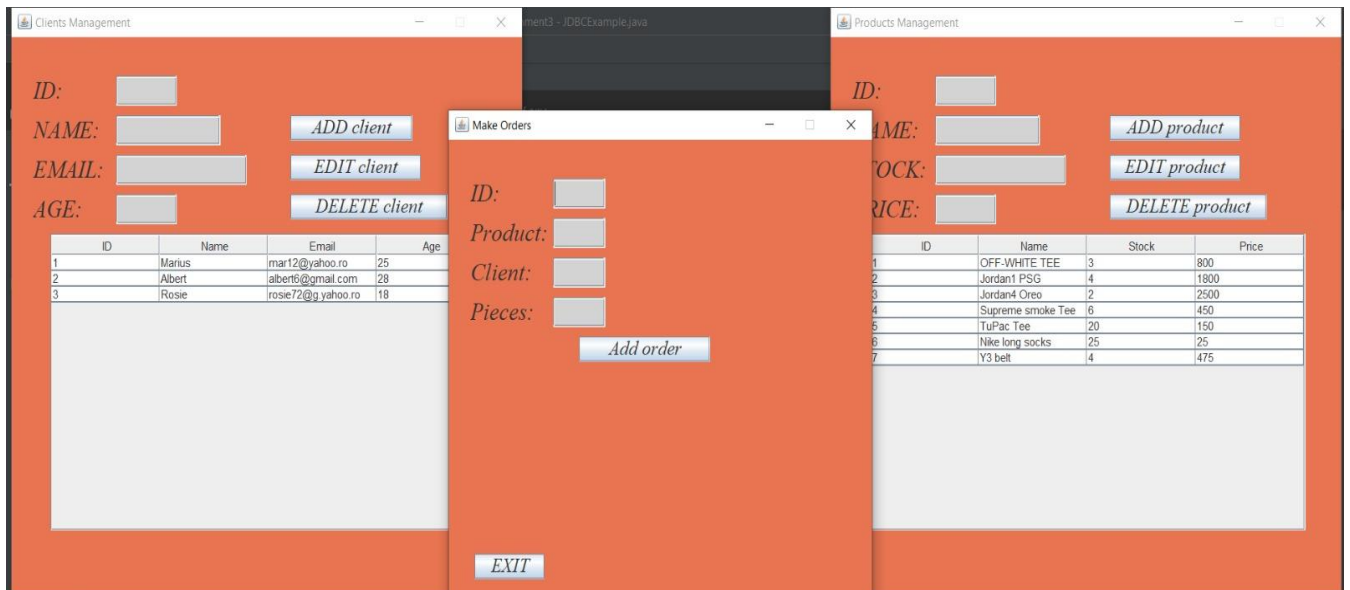


Am folosit relații de 1 to 1 în WorkBench, MySQL.

4. Implementare

În cadrul pachetului "Start" avem clasa main, care pornește programul.

Mai apoi, în pachetul "Presentation" se află clasa View. Cu ajutorul acestei clase apare pe ecran o pagină care ne permite să continuăm mai departe rularea programului sau să abandonăm. În cazul în care se dorește continuarea programului, pe ecran vor apărea 3 frame-uri diferite. Unul pentru operații de tip CRUD pe tabela "clients", alt frame pentru același tip de operații pe tabela "products" și crearea unei noi comenzi. Interfața este una simplă, fiind ușor de utilizat pentru orice user, chiar dacă acesta nu este foarte familiar cu lucruri de genul.



În cadrul pachetului, "Model" avem clasele principale care sunt folosite de majoritatea celorlalte clase. Fiecare tabelă din acest pachet conține doar constructori și gettere/settere, mai puțin clasa "Bill" care este de tip record pentru clasa "Order".

```
package Model;

/**
 * In this class we take note of the orders created
 *
 * @author Elecfi Sergiu - Andrei 30223
 * @date 25.05.2023
 */
@MonkeYzPGL
public record Bill(int orderId, int price) {
}
```

De asemenea, în cadrul pachetului "BLL" avem clasele care apelează metodele create în pachetul "Data Access", și se ocupă de validarea datelor.

```
public class ClientBLL {  
    3 usages  
    private List<Client> clienti;  
    5 usages  
    private ClientDAO CL;  
    MonkeYzPGL  
    public ClientBLL(){...}  
  
    MonkeYzPGL  
    public List<Client> getAll(){...}  
  
    MonkeYzPGL  
    public void insert(Client client) { CL.insert(client); }  
  
    MonkeYzPGL  
    public void delete(Client client) { CL.delete(client); }  
  
    MonkeYzPGL  
    public void update(Client client) { CL.update(client); }  
}
```

Celelalte două clase au ambele aceeași idee de programare la mijloc, doar numele metodelor diferind și apelul acelor.

În interiorul pachetului "Data Access" avem implementate metodele de tip CRUD, dar și o metodă care returnează întregul conținut al tabelului cerute. Această clasă a fost implementată cu ajutorul reflexiei, pentru a nu mai fii nevoie să se scrie cod separat pentru fiecare tabelă în cazul în care dorim să afișăm datele din ele, sau să le verificăm.

Codul pentru această metodă este :

```
public List<T> getAll(){
    List<T> sir = new ArrayList<>();
    String query = "SELECT * FROM " + table;
    Connection conn = null;
    PreparedStatement statement = null;
    ResultSet rs = null;
    try{
        conn = ConnectionFactory.getConnection();
        statement = conn.prepareStatement(query);
        rs = statement.executeQuery();
        ResultSetMetaData metaData = rs.getMetaData();
        int col = metaData.getColumnCount();
        while(rs.next()){
            T x = createInstance();
            for(int i = 1; i <= col; i++){
                String colName = metaData.getColumnName(i);
                Object colValue = rs.getObject(i);
                setFieldValue(x,colName,colValue);
            }
            sir.add(x);
        }
    } catch(SQLException e){
```

Celelalte clase din acest pachet implementează operațiile basic pe baza de date, însă interogările se fac în codul scris în Java, nefiind nevoie să se creeze proceduri separate în WorkBench, MySQL.

```
String s = "INSERT INTO clients (id, name, email, age) VALUES (?, ?, ?, ?)";
Connection conn = null;
PreparedStatement statement = null;
ResultSet rs = null;
try {
    conn = ConnectionFactory.getConnection();
    statement = conn.prepareStatement(s);

    statement.setInt( parameterIndex: 1, client.getId());
    statement.setString( parameterIndex: 2, client.getName());
    statement.setString( parameterIndex: 3, client.getEmail());
    statement.setInt( parameterIndex: 4, client.getAge());

    statement.executeUpdate();
    System.out.println("Insertion successful");
}
```


În interiorul clasei "OrderDAO" se creează o comandă și se inserează în interiorul tabelii respective din baza noastră de date, comanda nouă creată. De asemenea, din stock-ul produsului comandat se scade numărul de bucăți comandate, astfel că stock-ul este mereu actualizat în timp real.

```
int rowsAffected = preparedStatement.executeUpdate();
if (rowsAffected > 0) {
    String updateQuery = "UPDATE products SET stock = stock - ? WHERE id = ?";
    preparedStatement = conn.prepareStatement(updateQuery);
    preparedStatement.setInt( parameterIndex: 1, order.getNr());
    preparedStatement.setInt( parameterIndex: 2, order.getIdProduct());
    preparedStatement.executeUpdate();
}
```

În final, în interiorul pachetului "Connection" se află o singură clasă "ConnectionFactory", care a face conexiunea cu baza de date la fiecare apelare a acesteia, și care de asemenea încheie conexiunea dintre IDEA-ul folosit și WorkBench prin apelarea metodelor close.

```
1 usage  👤 MonkeYzPGL
private Connection createConnection(){...}

👤 MonkeYzPGL
public static Connection getConnection() { return singleInstance.createConnection(); }

👤 MonkeYzPGL
public static void close(Connection connection) {...}

👤 MonkeYzPGL
public static void close(Statement statement){...}

👤 MonkeYzPGL
public static void close(ResultSet resultSet){...}
```


5. Rezultate

Proiectul a fost testat pentru toate condițiile de utilizare. În cazul în care acțiunea dorită s-a putut efectua în linia de comandă al IDE-ului vor apărea mesajele respectivei acțiuni. În caz contrar vor apărea sub culoare roșie mesajele erorilor apărute.

Proiectul a fost testat pe toate operațiile de tip CRUD, de creare a comenzilor. S-a verificat și clasa Log, unde apar "chitanțele" pentru fiecare comandă.

Proiectul este 100% funcțional. Toate erorile care ar fi posibil să apară au fost tratate.

6. Concluzii

Pe parcursul realizării proiectului am învățat ce înseamnă principiul de "reflexie" în Java și mi-am aprofundat lucrul cu Java combinat cu MySQL.

Dificultățile întâlnite au apărut la reflexie deoarece este un principiu nou, și nu ușor de implementat, dar cu ajutorul materialelor primite în cadrul laboratorului și unele surse și videoclipuri de pe internet am reușit să realizez și această cerință.

Viitoare implementări ale programului ar putea fi utilizarea reflexiei în realizarea operațiilor de tip CRUD, fără a mai fi nevoie să se scrie cod separat pentru fiecare clasă sau tabelă în parte.

Consider că acest proiect este unul care cel puțin pe mine mă va ajuta în viitor, dar probabil că pe majoritatea îi va ajuta deoarece necesită o logică a creării unui bussines.

7. Bibliografie

1. https://gitlab.com/utcn_dsrl/pt-layered-architecture
2. <https://www.baeldung.com/javadoc>
3. <https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html>
4. <https://jenkov.com/tutorials/java-reflection/index.html>
5. <https://mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/>
6. Cursurile de la materia din acest semestru "Tehnici de programare"
7. Materiile din semestrul trecut "Proiectare orientată pe obiecte" și "Baze de date".

